ChatGPT

Process Enumeration in Windows – `EnumProcesses` and Code Walkthrough

The Windows API function `EnumProcesses` (in `Psapi.h`) retrieves the process identifier (PID) for **each** running process in the system [1]. In other words, it produces a list of all active process IDs, similar to writing down the nameplate of every office in a building. An analogy: imagine walking through a large office building and noting down the employee IDs (process IDs) outside every office door (process). That list of IDs is what `EnumProcesses` provides. The code then examines each ID to find a matching process name.

Using analogies, the code's overall flow is like a security guard checking who is in each office: it lists all office numbers (`EnumProcesses`), then for each office it "opens" the door (`OpenProcess`), checks the person's name badge (`GetModuleBaseName`), and compares it to the target name (e.g. `"svchost.exe"`). If it finds a match, it reports the office number (PID) and stops. This is useful in a defensive context to understand how malware might perform reconnaissance – by enumerating processes to find a specific service or application – and to ensure we know how to monitor such behavior. (In fact, attackers often use `EnumProcesses` to scan a system as a first step [2].)

How the Code Works (Step-by-Step)

The code defines a function `GetRemoteProcessHandle(szProcName, &Pid, &hProcess)` that **finds the PID and handle of a target process name**. Here are the main steps of the algorithm:

1. **Call `EnumProcesses` to list all PIDs.** A large DWORD array is allocated (e.g. 2048 entries) and passed to `EnumProcesses`, which fills it with all active PIDs [1]. On return, `dwReturnLen1` tells us how many bytes were written; dividing by `sizeof(DWORD)` gives the number of PIDs.

2. **Loop over each PID.** The code iterates `i` from 0 to `dwNmbrOfPids-1`. For each PID in `adwProcesses[i]`:

3. If the PID is nonzero, attempt to open the process with `OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid)` [3]. This returns a **handle** to the process if successful, or `NULL` if it fails (e.g. due to permissions) [4].
4. If `OpenProcess` returns a valid handle `hProcess`, the code calls `EnumProcessModules(hProcess, &hModule, sizeof(hModule), &dwReturnLen2)` [5]. This API returns one or more module handles for that process. In this code, only the first module handle (`hModule`) is retrieved (which is typically the main executable).
5. If `EnumProcessModules` succeeds, the code calls `GetModuleBaseName(hProcess, hModule, szProc, MAX_PATH)` [6]. This fills `szProc` with the **name of the module** (the process executable name).

6. Compare the retrieved name (`szProc`) to the target name `szProcName` (e.g. `"svchost.exe"`) using `wcscmp`. If they match, we have found the target process. The code stores the PID and handle, prints a found message, and breaks out of the loop.
7. Whether or not the name matches, it then calls `CloseHandle(hProcess)` [7] to release the process handle (if it was opened) before moving on (unless it already broke out).

8. **Return result.** After the loop, the function returns `TRUE` if a matching process was found (and its PID/handle set), or `FALSE` otherwise. The `main` function then prints the result.

In pseudo-code, the core loop is:

- Get array of all PIDs with `EnumProcesses`.
- For each PID in the array:
- If PID is not 0:
  - Try `OpenProcess` to get a handle. If `NULL`, skip.
  - Call `EnumProcessModules` on the handle. If it fails, skip.
  - Call `GetModuleBaseName` on the handle/module to get the process name.
  - **If** the name equals the target, save PID/handle and exit.
  - Close the handle (cleanup).
- End loop and return whether found.

The code also includes a `PrintProcesses()` helper (commented out) that does a similar enumeration but prints every process name and PID. It calls the same sequence: `EnumProcesses`, then for each PID it opens the process (with query/read rights) and uses `EnumProcessModules` + `GetModuleBaseName` to retrieve and print each name [5] [6].

Line-by-Line Explanation

- **Headers and macro:** The code includes `<Windows.h>` and `<Psapi.h>` to use these API functions. `TARGET_PROCESS` is defined as the target executable name (e.g. `L"svchost.exe"`).

- **EnumProcesses call:** In `GetRemoteProcessHandle`, we declare a large `DWORD adwProcesses[2048]`. The call

  ```
  EnumProcesses(adwProcesses, sizeof(adwProcesses), &dwReturnLen1)
  ```

  fills `adwProcesses` with all PIDs. If it fails, the code prints an error (using `GetLastError()`) [4] [8] and returns false. Otherwise, `dwReturnLen1/sizeof(DWORD)` gives the count of PIDs.

- **For loop:** We loop `for (int i = 0; i < dwNmbrOfPids; i++)`. Inside:

- `if (adwProcesses[i] != 0)`: Skip PID 0 (Idle process).
- `hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, adwProcesses[i])`: Attempts to open every process with full access [3]. (In `PrintProcesses`, it uses lesser rights `PROCESS_QUERY_INFORMATION | PROCESS_VM_READ` for safety.)

- If `hProcess != NULL`, we got a valid handle. We then do:
    - `EnumProcessModules(hProcess, &hModule, sizeof(hModule), &dwReturnLen2)`. This returns at least one module handle (typically the EXE) [5]. If this fails, we print an error and skip.
    - On success, call `GetModuleBaseName(hProcess, hModule, szProc, MAX_PATH)`. This writes the module's base name (the executable name) into `szProc` [6]. If this fails, print an error.
    - Compare `szProc` with `szProcName` (e.g. `"svchost.exe"`) using `wcscmp`. If they match, we found the target. The code then sets the output PID and handle (`*pdwPid = adwProcesses[i]; *phProcess = hProcess;`) and breaks the loop.

- If the name didn't match, or if opening/enum failed, the code closes the handle with `CloseHandle(hProcess)` [7] before continuing.

- **Return values:** After the loop, the code checks if a PID/handle was found (`*pdwPid` and `*phProcess` are non-null) and returns `TRUE` or `FALSE` accordingly. The `main()` function then prints the found PID and waits for Enter.

Key points:
- `EnumProcesses` simply lists PIDs [1]; it does not give names.
- `OpenProcess` with sufficient rights gets a handle we can inspect [3].
- `EnumProcessModules` and `GetModuleBaseName` together allow us to get the executable name from that process [5] [6].
- `CloseHandle` cleans up each process handle when done [7].
- Throughout, `GetLastError()` (not shown in detail) is used whenever a WinAPI fails to retrieve the specific failure reason [4] [8].

Example Algorithm Steps

To make the logic clear, here is a summary of the algorithm in numbered steps:

1. **Retrieve all PIDs**: Call `EnumProcesses`, storing results in `adwProcesses[]`. Compute count: `dwNmbrOfPids = dwReturnLen1 / sizeof(DWORD)`.
2. **Loop through PIDs**: For each PID in the array:
   a. If PID is nonzero, attempt `hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid)`. If `hProcess` is NULL, skip this PID.
   b. Call `EnumProcessModules(hProcess, &hModule, sizeof(HMODULE), &dwReturnLen2)`. If it fails, close handle and continue.
   c. Call `GetModuleBaseName(hProcess, hModule, szProc, MAX_PATH)` to get the process name string.
   d. If `szProc` equals the target name (e.g. `"svchost.exe"`), store `pid` and `hProcess` as the result and break out of loop. Otherwise, call `CloseHandle(hProcess)` and continue.
3. **Finish**: After the loop, return TRUE if a process was found, or FALSE if not.

The `PrintProcesses` function similarly enumerates PIDs and prints each process name and ID; it uses the same calls (`EnumProcesses`, `OpenProcess`, `EnumProcessModules`, `GetModuleBaseName`, and `CloseHandle`) to obtain the name of each process for display [5] [6].

Windows APIs Used

The key Windows API functions in this code are:

- `EnumProcesses` (in *Psapi.lib*, *Kernel32.dll*) – enumerates all running process IDs [1].
- `OpenProcess` (in *Kernel32.dll*) – opens a handle to a given PID with specified access rights [3].
- `EnumProcessModules` (in *Psapi.lib*, *Kernel32.dll*) – retrieves module handles for a process handle (typically returning the main executable module) [5].
- `GetModuleBaseName` (in *Psapi.lib*, *Kernel32.dll*) – gets the base name of a module (file name) given a process handle and module handle [6].
- `CloseHandle` (in *Kernel32.dll*) – closes an open object handle (here used to release process handles) [7].
- `GetLastError` (in *Kernel32.dll*) – retrieves the calling thread's last-error code, used for error reporting when API calls fail [4] [8].

These API calls are the standard way in Windows to enumerate and inspect processes. Understanding them is useful both for defense (e.g. monitoring for suspicious use of these calls) and for offense (malware often uses them to find target processes) [2] [9].

Sources: Microsoft documentation and Windows developer references describe each function's behavior [1] [3] [5] [6] [7]. These explain how `EnumProcesses` returns PIDs, how to open and inspect each process, and how to retrieve module names. The code's logic simply chains these calls to locate the target process by name.

---

[1] EnumProcesses function (psapi.h) - Win32 apps | Microsoft Learn
https://learn.microsoft.com/en-us/windows/win32/api/psapi/nf-psapi-enumprocesses

[2] Find Processes with EnumProcesses | by S12 - H4CK | Medium
https://medium.com/@s12deff/find-processes-with-enumprocesses-52ef3c07446a

[3] [4] OpenProcess function (processthreadsapi.h) - Win32 apps | Microsoft Learn
https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-openprocess

[5] EnumProcessModules function (psapi.h) - Win32 apps | Microsoft Learn
https://learn.microsoft.com/en-us/windows/win32/api/psapi/nf-psapi-enumprocessmodules

[6] GetModuleBaseNameA function (psapi.h) - Win32 apps | Microsoft Learn
https://learn.microsoft.com/en-us/windows/win32/api/psapi/nf-psapi-getmodulebasenamea

[7] CloseHandle function (handleapi.h) - Win32 apps | Microsoft Learn
https://learn.microsoft.com/en-us/windows/win32/api/handleapi/nf-handleapi-closehandle

[8] GetLastError function (errhandlingapi.h) - Win32 apps | Microsoft Learn
https://learn.microsoft.com/en-us/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror

[9] Detecting Running Process: EnumProcess API - Unprotect Project
https://unprotect.it/technique/detecting-running-process-enumprocess-api/